**MIPS Instruction Coding**

**Instruction Coding Formats**

MIPS instructions are classified into four groups according to 3 coding formats , NO Pseudo-instructions or co-processor

instructions required in Assignment1&2 .

 R-Type - This group contains all instructions that do not require an immediate value, target offset, memory address displacement, or memory address to specify an operand. This includes arithmetic and logic with all operands in

registers, shift instructions, and register direct jump instructions (jal r and jr) .

All R-type instructions use opcode 000000.

 I-Type - This group includes instructions with an immediate operand, branch instructions, and load and store

instructions. In the MIPS architecture, all memory accesses are handled by the main processor, so coprocessor load and store instructions are included in this group.

All opcodes except 000000, 00001x, and 0100xx are used for I-type instructions.

 J-Type - This group consists of the two direct jump instructions (j and jal). These instructions require a memory address to specify their operand.

J-type instructions use opcodes 00001x.

 Coprocessor Instructions - MIPS processors all have two standard coprocessors, CP0 and CP1. CP0 processes

various kinds of program exceptions. CP1 is a floating point processor. The MIPS architecture makes allowance

for future inclusion of two additional coprocessors, CP2 and CP3. All coprocessor instructions use opcodes 0100xx.

Note: ALL arithmetic immediate values are sign-extended. After that, they are handled as signed or unsigned 32-bit numbers, depending upon the instruction. Signed instructions can generate an overflow exception; unsigned cannot.

**R-Type Instructions (Opcode 000000)**

Main processor instructions that do not require a target address, immediate value, or branch displacement use an R-type

<http://www.cs.sunysb.edu/>~lw/spim/MIPSinstHex.pdf Hex extension by Larry Wittie to original instruction lists by Gary Shute, 26-Jun-2007 Page 1 of 4

MIPS Instruction Coding

2/19/09 1:56 PM

coding format. This format has fields for specifying of up to three registers and a shift amount. For instructions that do not use all of these fields, the unused fields are coded with all 0 bits. All R-type instructions use a 000000 opcode. The operation is specified by the function field.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode (6) | rs (5) | rt (5) | rd (5) | sa (5) | function (6) |

**Alf Instruction Function**

add rd, rs, rt 100000 addu rd, rs, rt 100001

**Opcd Funct Description Numeric Instruction Function Funct Hex**

0x00 0x20 Add (with overflow)

0x00 0x21 Add unsigned (no overflow)

|  |  |  |
| --- | --- | --- |
| and rd, rs, rt 100100  div rs, rt 011010 | 0x00 0x24 Bitwise and  (for debugging)  0x00 0x1A Divide | **Please note that the 'break' instructions is deleted** |
| divu rs, rt 011011 | 0x00 0x1B Divide unsigned |  |
| jalr rd, rs 001001 | 0x00 0x09 Jump and link |  |
| jr rs 001000 | 0x00 0x08 Jump register |  |
| mfhi rd 010000 | 0x00 0x10 Move from HI |  |
| mflo rd 010010 | 0x00 0x12 Move from LO |  |
| mthi rs 010001 | 0x00 0x11 Move to HI |  |
| mtlo rs 010011 | 0x00 0x13 Move to LO |  |
| mult rs, rt 011000 | 0x00 0x18 Multiply |  |
| multu rs, rt 011001 | 0x00 0x19 Multiply unsigned |  |
| nor rd, rs, rt 100111 | 0x00 0x27 Bitwise nor |  |
| or rd, rs, rt 100101 | 0x00 0x25 Bitwise or |  |
| sll rd, rt, sa 000000 | 0x00 0x00 Shift left logical |  |
| sllv rd, rt, rs 000100 | 0x00 0x04 Shift left logical variable |  |
| slt rd, rs, rt 101010 | 0x00 0x2A Set on less than (signed) |  |
| sltu rd, rs, rt 101011 | 0x00 0x2B Set on less than unsigned |  |
| sra rd, rt, sa 000011 | 0x00 0x03 Shift right arithmetic |  |
| srav rd, rt, rs 000111 | 0x00 0x07 Shift right arithmetic variable |  |
| srl rd, rt, sa 000010 | 0x00 0x02 Shift right logical |  |
| srlv rd, rt, rs 000110 | 0x00 0x06 Shift right logical variable |  |
| sub rd, rs, rt 100010 | 0x00 0x22 Subtract |  |
| subu rd, rs, rt 100011 | 0x00 0x23 Subtract unsigned |  |
| syscall 001100 | 0x00 0x0C System call |  |
| xor rd, rs, rt 100110 | 0x00 0x26 Bitwise exclusive or |  |

**I-Type Instructions (All opcodes except 000000, 00001x, and 0100xx)**

I-type instructions have a 16-bit immediate field that codes an immediate operand, a branch target offset, or a

displacement for a memory operand. For a branch target offset, the immediate field contains the signed difference

between the address of the following instruction and the target label, with the two low order bits dropped. The dropped bits are always 0 since instructions are word-aligned.

<http://www.cs.sunysb.edu/>~lw/spim/MIPSinstHex.pdf

Page 2 of 4

MIPS Instruction Coding 2/19/09 1:56 PM

For the bgez, bgtz, blez, and bltz instructions, the rt field is used as an extension of the opcode field.

opcode (6) rs (5) rt (5) immediate (16)

**Alf Instruction Opcode Notes Opcd Description**

addi rt, rs, immediate 001000 addiu rt, rs, immediate 001001 andi rt, rs, immediate 001100 beq rs, rt, label 000100 bgez rs, label 000001

bgtz rs, label 000111

blez rs, label 000110 bltz rs, label 000001 bne rs, rt, label 000101 lb rt, immediate(rs) 100000 lbu rt, immediate(rs) 100100 lh rt, immediate(rs) 100001 lhu rt, immediate(rs) 100101 lui rt, immediate 001111 lw rt, immediate(rs) 100011

0x08 Add immediate (with overflow)

0x09 Add immediate unsigned (no overflow)

0x0C Bitwise and immediate

0x04 Branch on equal

rt=00001 0x01 Branch on greater than or equal to zero

rt=00000 0x07 Branch on greater than zero

rt=00000 0x06 Branch on less than or equal to zero

rt=00000 0x01 Branch on less than zero

0x05 Branch on not equal

0x20 Load byte

0x24 Load byte unsigned

0x21 Load halfword

0x25 Load halfword unsigned

0x0F Load upper immediate

0x23 Load word

lwc1 rt, immediate(rs) 110001 0x31 Load word coprocessor 1

|  |  |
| --- | --- |
| ori rt, rs, immediate 001101 | 0x0D Bitwise or immediate |
| sb rt, immediate(rs) 101000 | 0x28 Store byte |
| slti rt, rs, immediate 001010 | 0x0A Set on less than immediate (signed) |
| sltiu rt, rs, immediate 001011 | 0x0B Set on less than immediate unsigned |
| sh rt, immediate(rs) 101001 | 0x29 Store halfword |
| sw rt, immediate(rs) 101011 | 0x2B Store word |

swc1 rt, immediate(rs) 111001 0x39 Store word coprocessor 1

|  |  |  |
| --- | --- | --- |
| xori rt, rs, immediate 001110 | 0x0E Bitwise exclusive or immediate |  |
| lwl rt, immediate(rs) 100010 | 0x22 Load word left (unaligned) | **Please note that the last 4** |
| lwr rt, immediate(rs) 100110 | 0x26 Load word right (unaligned) | **instructions are updated** |
| swl rt, immediate(rs) 101010 | 0x2A Store word left (unaligned) | **(lwl, lwr, swl, swr)** |
| swr rt, immediate(rs) 101110 | 0x2E Store word right (unaligned) |  |

**J-Type Instructions (Opcode 00001x)**

The only J-type instructions are the jump instructions j and jal. These intructions require a 26-bit coded address field to specify the target of the jump. The coded address is formed from the bits at positions 27 to 2 in the binary representation of the address. The bits at positions 1 and 0 are always 0 since instructions are word-aligned.

When a J-type instruction is executed, a full 32-bit jump target address is formed by concatenating the high order four bits of the PC (the address of the instruction following the jump), the 26 bits of the target field, and two 0 bits.

MIPS Instruction Coding

2/19/09 1:56 PM

|  |  |
| --- | --- |
| opcode (6) | target (26) |

**Instruction Opcode Target Opcd Description**

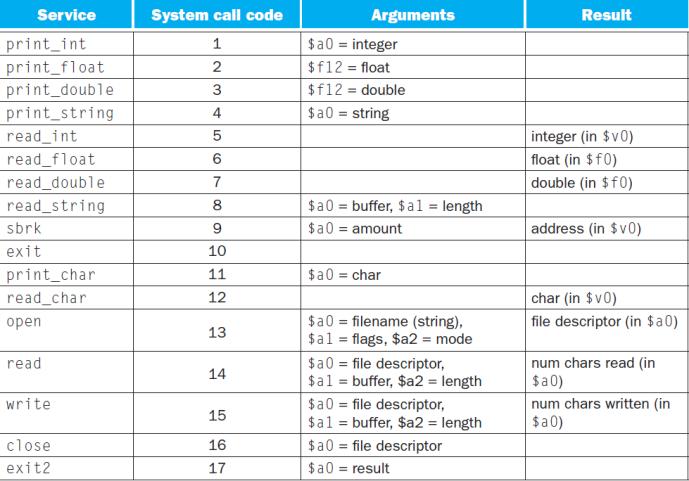
j label 000010 coded address of label 0x02 Jump

jal label 000011 coded address of label 0x03 Jump and link

**Syscalls you need to support in Project2(not required in project1)**

The syscalls you need to support are: 1, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17 in the following

chart.



**Page URL:** <http://www.cs.sunysb.edu/>~lw/spim/MIPSinstHex.pdf from <http://www.d.umn.edu/>~gshute/spimsal/talref.html **Page Author:** extensions by Larry Wittie from original no-hex instruction lists by Gary Shute

**Last Modified:** Saturday, 18-Sep-2010 from original of Tuesday, 26-Jun-2007 12:33:40 CDT

**Comments to:** lw AT ic DOT sunysb DOT edu (Original instruction list author was gshute@d.umn.edu)

<http://www.cs.sunysb.edu/>~lw/spim/MIPSinstHex.pdf Page 4 of 4

**How syscall works?**

Recall that you will read in a MIPS file as input in your project 1, and you will need to run the code (Read Project 1 instruction first if you do not understand). In your assembling part, you simply need to put the binary code:

00000000000000000000000000001100 in, all syscalls have the same machine code.

How do we distinguish them then?Syscalls are distinguished by checking the value stored in $v0 register. In your simulation part, when ever you see a syscall made (seeing this binary code: 00000000000000000000000000001100), you simply go and check

what is stored in $v0. In other words, you can write a switch, with the value in $v0 being the cases. For each case, you simply

implement its functionality. Using print\_int as an example, in your case: 1 , you can have printf("%d", \*a0); . The argument column specifies the arguments this syscall takes (the integer to be printed is stored in $a0,for example).

**The input MIPS code format**

You will need to consider the following situations while reading the input MIPS file:

1. **There will only be .data and .text sections.**

2. There could be spaces or tabs before and after each line.

3. There could be spaces before and after each element within a line. e.g. add $t0, $t1, $t2 .

4. There could be empty lines.

5. There could be comments after the line of code. There could also be a line with only comments.

Comments are always following a "#".

6. Labels can be followed by a line of code, or can have it's own line. Labels are labeling the same line of code in both situations.

|  |
| --- |
| case1  label: add $t0, $t1, $t2  case2  label:  add $t0, $t1, $t2 |

**The data types you need to support (for project 2)**

The data types you need to support are:

1. ascii

2. asciiz

3. word

4. byte

5. half